UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/042,079	01/07/2002	Scott J. Broussard	AUS920010996US1	6556

35525 7590 04/20/2006

IBM CORP (YA)
C/O YEE & ASSOCIATES PC
P.O. BOX 802333
DALLAS, TX 75380

EXAMINER	
ROMANO, JOHN J	
ART UNIT	**PAPER NUMBER**
2192	

DATE MAILED: 04/20/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

MAILED

APR 20 2006

Technology Center 2100

Application Number: 10/042,079
Filing Date: January 07, 2002
Appellant(s): BROUSSARD, SCOTT J.

Peter B. Manzo
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed 01/06/2006 appealing from the Office action mailed 8/26/2005.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is substantially correct – See item (9) for correct statement of grounds of rejection.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

2003/0051233 A1	Krishna et al.	03-2003
6,836,884 B1	Evans et al.	12-2004
2002/0147763 A1	Lee et al.	10-2002
5,590,331	Lewis et al.	12-1996

Dale Green, "Trail: The Reflection API", The Java Tutorial. Posted November 27th, 1999. Retrieved from http://java.sun.com/books/tutorial/reflect/, pp.1-36

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

1. Claims 1, 7, 10-14, 20, 23-27, 33 and 36-39 are rejected under 35. U.S.C. §102(e) as being anticipated by Krishna et al., US 2003/0051233 (hereinafter **Krishna**).

2. Claims 2, 4-6, 9, 15, 17-19, 22, 28, 30-32 and 35 are rejected under 35 U.S.C. §103(a) as being unpatentable over Krishna et al., United States Patent Publication No. 2003/0051233 A1 (hereinafter "Krishna"), in view of Dale Green, "Trail: The Reflection API, The Java Tutorial, November 27th, 1999, (hereinafter "Green").

3. Claims 3, 16 and 29 are rejected under 35 U.S.C. §103(a) as being unpatentable over Krishna et al., United States Patent Publication No. 2003/0051233 A1 (hereinafter "Krishna"), in view of Dale Green, "Trail: The Reflection API, The Java Tutorial, November 27th, 1999, (hereinafter "Green") and further in view of Evans et al., US 6,836,884 B1 (hereinafter Evans).

4. Claims 8, 21 and 34 are rejected under 35 U.S.C. §103(a) as being unpatentable over Krishna et al., United States Patent Publication No. 2003/0051233 A1 (hereinafter "Krishna"), in view of Dale Green, "Trail: The Reflection API, The Java Tutorial, November 27th, 1999, (hereinafter "Green").

(10) Response to Argument

Appellants arguments filed January 06th, 2006, have been fully considered but they are not persuasive. For example,

(A) Firstly, the following is a rejection of claim 1, which maps the rejections and is used in the Examiner's answers to the arguments:

In regard to claim **1**, **Krishna** discloses:

- "*A method in a data processing system for generating a generic compilation interface from a first object-oriented software package, said method comprising the steps of...*" (E.g., see Figure 3 & Page 2, Paragraph [0032]), wherein the secondary developer combines the developers own source code 160, with the equivalent public classes (library stubs

220), compiles it with java compiler 125B, thereby generating a compilation interface 170, for first package 160, wherein the interface 170, receives the compiled source code 160, from the first package and interfaces with the card converter 335B.

- "*...for each of said public classes, identifying all public entities included in each of said public classes...*" (E.g., see Figure 7A & 7B & Page 2, Paragraph [0025]), wherein each public class is identified (Figure 7A, line 709) and each public within the identified public class is identified (Figure 7B, method, lines 719-720), wherein the classes are included by reference from the source code 160 and the class declaration and interface (input requirements) are provided to the compiler 125B, via the stubs with the references to the executable source code previously contained in the method body excluded as taught below.
- "*...removing all references to software that is defined in a second software package from said public entities included in each of said public classes ...*" (E.g., see Figure 7B & Page 2, Paragraph [0025]), wherein during the process of generating the library stubs 220, the executable source code is removed from the method and replaced with generic values, wherein if the method (entity) returns a reference, the reference is

replaced with a null value or corresponding type (Figure 7B, lines 726-729), resulting in the references to the software (executable statements) contained in the public entity (method) being replaced (removed), by replacing the references to the source code 110, with generic stubs 220, thereby removing all references to software defined in public entity in public classes defined in the second software package (110) and included in the first software package (160).

- "*...generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second package...*" (E.g., see Figure 7B & Page 4, Paragraph [0049]), wherein the library stub generator (Figure 3, 315), processes each methods (all public entities) included in the class by generating a stub, wherein Figure 7B, steps 721- 734, teach replacing a reference to software contained in the body of a method of the source code 110, with a stub returned with the appropriate return type value. It should also be noted that this method is performed for all non-private (public) classes. Additionally, any exception is processed in another file, but nonetheless processed, so that the secondary developer can compile class files.

- "*...compiling each of said equivalent public classes; and generating a compilation interface for said first package including each of said compiled equivalent public classes.*" (E.g., see Figure 3 & Page 2, Paragraph [0032]), wherein the secondary developer combines the developers own source code 160, with the equivalent public classes (library stubs 220), compiles it with java compiler 125B, thereby generating a compilation interface 170, for first package 160, wherein the interface 170, receives the compiled source code 160, from the first package and interfaces with the card converter 335B.

It should be noted that Krishna is used to reject the entire claim 1 and therefore, the arguments in regard to motivation are moot.

It should also be noted that the generic stubs 220, taught by *Krishna*, are the same as Applicant's generic interface, wherein both take the name and function declaration and replace the return value with a default value according to the type in order to bypass compiler reference errors (to the otherwise required source code), wherein the required source code is not needed, instead the generic stubs are implemented (see below). *Krishna's* stubs are the equivalent to the definition of Applicant's generic interface (See original specification, Page 16, lines 24-26) as per the specification, wherein the return values are substituted with generic values. This is disclosed by *Krishna* in Figure 7B, steps 726-729, wherein during the process of generating the stubs, the substitute return values are set if the method returns a reference. In other words the reference is replaced with the generic return value in order to eliminate the references along with all other executable source code. The interface is generic because the stubs replace the actual source code with generic default code, based on the expected return type (See Krishna, Table 7B, steps 724-729) as addressed above.

As a preliminary note, the Examiner agrees with Applicant's statement that *Krishna* teaches a method of Java Card application development in order to protect the first developer's intellectual property rights in source code (page 12, second paragraph of the appeal brief). The Examiner also agrees with Applicant's statement the *Krishna* teaches that the cross-referenced source code must be included within the Java Card application in order to be suitable for interpretation by the Java Virtual Machine (brief, page 12, third paragraph). However, Applicant then concludes that this is in contrast to the present invention (brief, Page 12, last paragraph – Page 13, first paragraph), seemingly based on the argument that "...in order to be suitable for interpretation by the Java Virtual Machine, a Java Card executable must include all elements necessary for on-card interpretation, such as cross-references to code outside the executable itself", which the Examiner respectfully, but strongly disagrees. The portion of the reference cited in the above passage by Applicant is referring to the java Card (See Figure 3, 152) which takes place after the cited portions of prior art and has nothing to do with the claim rejections. The claim rejections involve generating a generic compilation interface not implementing executable code on a Java card. The fact that executable code may later be provided for running on a Java Card 152, has nothing to do with the claim rejections. Thus, the argument is moot and the rejection is maintained.

In response to the argument that Krishna does not teach or suggest "removing all references, or cross-references, to any other software applications within the generated compilation interface as recited in claim 1" (brief, page 13, first paragraph), the Examiner disagrees. *Krishna* teaches (E.g., see Figure 3), replacing the referenced source code 110 (second source code), with stubs 315 as addressed above, thereby excluding the referenced source code in the body of the public entity that would be otherwise required without the stubs as disclosed in *Krishna's* method. Figure 1, shows the prior art method, wherein the source code 110, would be required as input into the Java Compiler 125B. However, Krishna's method excludes the Java Source code 110 public entities, as shown in figure 3,7A and 7B, and replaces it with Library stubs 220.

For example, *Krishna* discloses:

- "The library stubs 220 <u>exclude the source code 110 executable statements</u>, but includes declarations and interfaces of source code 110, so that the secondary developer can compile (binary) class files 170, for converting to CAP files 180, etc.", emphasis added, (See *Krishna*, Page 2, Paragraph [0025] & Figure 7).

This is to say, *Krishna's* emphasized portion clearly discloses "removing (excluding) references to software (source code 110), that is defined in a second object-oriented software package" as cited in claim 1. From this perspective, (see Figure 2), the Java compiler 125B, receives library stubs 220, to exclude the source code 110 secondary software package) as disclosed above. Portions of the source code 110, such as declarations and interfaces are used to generate the stubs, however all references to the second software package, (source code 110) are excluded. This is evident by the source code 110, not being included.

In fact, Applicant also uses portions of the secondary software package to generate the interface. For example, Applicant discloses (Figure 4), wherein classes are enumerated (406), public entities are listed (410 + 412), and references are removed (414). Additionally, a return statement with a default value is provided (416) if necessary. The result is a generic compilation interface with the names and attributes of the original source code, which is equivalent to Krishna's stubs. All references to software in the original source code are replaced with the generic interface. Therefore including declarations in the stubs certainly does not mean that references are not removed as argued by Applicant (brief, page 13, second paragraph).

Additionally, references to interfaces are defined in an external software package from the source code 110, and therefore do not incorporate into the plain language of claim 1 as argued by Applicant (brief, page 13, second paragraph). Even arguably, it should be noted that if the references to these "interfaces" were not replaced within the stubs, *Krishna's* method would generate compile errors, and thus be ineffective. Therefore, Applicant's argument herein and Applicant's argument relating to cross references (page 13, third paragraph) are not persuasive as well as not included in the claims.

In regard to the argument that *Krishna* fails to mention anything about removing references to software that is defined in a second object-oriented software package (brief, page 13, third paragraph), see above. Further, in response to the argument that *Krishna* does not suggest the desirability of, or the motivation for, removing all other software application source code references (brief, page 13, third paragraph), the Examiner disagrees. Firstly, the argument is moot as the instant language "..all other source code..." is not included in the claim language. Furthermore, *Krishna* provides the following motivation:

- "Since the library source code could contain some very sensitive information, distributing this code could pose unacceptable risks." (Krishna, page 2, paragraph [0024]).

In response to the "unacceptable risks" involved in distributing the source code, *Krishna* teaches a method to exclude this source code as disclosed above. The library source code 110, (second object-oriented software package), is replaced with stubs, thereby replacing the reference to the library source code, or second object oriented software, with a stub comprising the declaration and interfaces of the source code (function name and return type); enabling the secondary developer to compile class files without an error. Thus, as disclosed herein, the references are resolved without providing the source code. Therefore, *Krishna* does indeed disclose motivation for removing or replacing referenced source code to overcome the "unacceptable risks" as addressed.

As per Applicant's argument that *Krishna* does not suggest the desirability of, or the motivation for, removing all other software applications source code reference (page 13, 3rd – 4th paragraphs of the appeal brief), the Examiner disagrees. As a preliminary note, Claim 1 is worded to remove all references to a second object-oriented software package as opposed to "...all other software applications source code references", as argued. Furthermore, the instant invention and *Krishna* may have been motivated by different factors, however, they both achieve the same objective of resolving references in order to compile without generating missing reference errors as disclosed by *Krishna* (paragraphs [0024] – [0025] and the instant application (original specification submitted with patent application, page 7, lines 11-15). Thus, the secondary developer can compile and develop source code for any reason, including without infringing on the intellectual property rights of an original library developer (E.g., see *Krishna*, Page 1, Paragraph [0006]). The fact that intellectual property rights happens to be mentioned in *Krishna's* method to resolve references, and not mentioned in Applicant's method, does not mean that *Krishna* does not teach removing all references to software that is defined in a second object-oriented software package, as disclosed above. In fact, *Krishna's* motivation is the same as the Applicant's, which is to resolve reference errors, as addressed above.

In regard to applicants' argument that removing all references to software that is defined in a second object-oriented software package to generate a generic compilation interface is distinguishable from manually removing executable source code to protect intellectual property rights of the first developer as taught by Krishna (brief, page 13, last paragraph – page 14, first paragraph), the Examiner disagrees, again based on the arguments above that Krishna does indeed teach removing all references. Additionally, Krishna does teach manually removing in Figure 2; however, Figure 3 teaches automatically removing the source code. Therefore, although not in the claims, manually removing references does not mean that Krishna's method does not read on the claim language of claim 1.

In regard to applicants' reference to Fig. 7A, and argument that *Krishna* teaches updating the integrated development environment to a class of objects not contained within the executable source code package (brief, page 14, first paragraph), the Examiner strongly disagrees. As shown above, *Krishna* clearly teaches removing references to software that is defined in a second object-oriented software package as stated in the claims, wherein the import list can be interpreted as a third software package.

Additionally, the import list is in regard to a second external file that may exist via parent-child relationships as the Java language allows a single inheritance model. It should be noted that this is interpreted as a <u>third object-oriented software package</u>, which is not addressed in claim 1, or any other claims. Furthermore, *Krishna* teaches updating the import list to include stubs, not references as alleged by the Applicant (page 14, first paragraph of the appeal brief). *Krishna* expressly teaches:

- "The export files 138 include an identifier mapping table of the previously mentioned symbol substitutions. The Java Card converter 135B <u>resolves</u> package 170 along with <u>the symbol references</u> in the export files138 and generates the secondary developer's applet CAP file180...", emphasis added, (Krishna, Paragraph [0026]).

This is to say that the symbol references are resolved by being replaced with the stubs or generic interface. Additionally, as noted above and illustrated in Figure 2, the interfaces are not provided to the Java compiler 125B and therefore, if they were not replaced with stubs as taught by Krishna, the method would not be functional.

Additionally, the return value is not a reference to software defined in a second software object-oriented software package as alleged by Applicant (brief, page 17 of 23, second paragraph). The return value is a replacement for a reference, thereby removing the reference to software that is defined in a second software package as in the application (Page 7, lines 2-4), wherein an original return statement includes a reference, and a default value is <u>substituted</u> for the reference, again as disclosed above.

In regard to applicants' reference to Fig. 7B, and related arguments (page 15, brief), the Examiner disagrees. The Examiner agrees with Applicant's statement that, "The return value is not a reference to software defined in a second software object-oriented software package (brief, page 15, second paragraph). The return value is a replacement for the reference as addressed above. Applicant then concludes, again, that "Krishna does not teach or suggest removing all references to software that is defined in a second object-oriented software package from the public entities included in each of the public classes...", (brief, page 16, first paragraph), seemingly based on the import list. In fact, the import list is set with information from the IDE file, which is then replaced by the stubs (or replacements for the references to software defined in a second software), thereby removing references to software that is defined in a object-oriented software package.

In regard to Applicants' argument that *Green* does not teach or suggest removing all references to software that is defined in a second object-oriented software package (brief, page 18 of 23, last paragraph), it is noted that *Green* is not relied upon in the previous office action to teach removing all references to software that is defined in a second object-oriented software package.

In view of the above arguments, independent claims 1, 14 and 27 are not allowable. Consequently, dependent claims 2, 4-7, 9-13, 15, 17-20, 22-26, 28, 30-33 and 35-39 are also not allowable, at least by virtue of their dependence on the independent claims.

(B) As shown in section A, above, Krishna teaches removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14 and 27. Therefore, the rejection of independent claims 1, 14 and 27 and dependent claims 3, 16 and 29 are maintained.

(C) As shown in section A, above, Krishna teaches removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14 and 27. Therefore, the rejection of independent claims 1, 14 and 27 and dependent claims 8, 21 and 34 are maintained.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

The following ground(s) of rejection are applicable to the appealed claims and were set forth in the Office Action mailed August 26th, 2005:

DETAILED ACTION

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1, 7, 10-14, 20, 23-27, 33 and 36-39 are rejected under 35 U.S.C. 102(e) as being anticipated by Krishna et al., US 2003/0051233 (hereinafter **Krishna**).

1. In regard to claims **1**, **14 and 27**, **Krishna** discloses:
   - "*A method in a data processing system for generating a generic compilation interface from a first object-oriented software package, said method comprising the steps of...*" (E.g., see Figure 3 & Page 2, Paragraph [0032]), wherein the secondary developer combines the developers own source code 160, with the equivalent public classes (library stubs 220), compiles it with java compiler 125B, thereby generating a compilation interface 170, for first package 160, wherein the interface 170, receives the compiled source code 160, from the first package and interfaces with the card converter 335B.

- "*...for each of said public classes, identifying all public entities included in each of said public classes...*" (E.g., see Figure 7A & 7B & Page 2, Paragraph [0025]), wherein each public class is identified (Figure 7A, line 709) and each public within the identified public class is identified (Figure 7B, method, lines 719-720), wherein the classes are included by reference from the source code 160 and the class declaration and interface (input requirements) are provided to the compiler 125B, via the stubs with the references to the executable source code previously contained in the method body excluded as taught below.
- "*...removing all references to software that is defined in a second software package from said public entities included in each of said public classes ...*" (E.g., see Figure 7B & Page 2, Paragraph [0025]), wherein during the process of generating the library stubs 220, the executable source code is removed from the method and replaced with generic values, wherein if the method (entity) returns a reference, the reference is replaced with a null value or corresponding type (Figure 7B, lines 726-729), resulting in the references to the software (executable statements) contained in the public entity (method) being replaced (removed), by replacing the references to the source code 110, with generic stubs 220, thereby removing all references to software defined in public entity in public classes defined in the second software package (110) and included in the first software package (160).
- "*...generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second package...*" (E.g., see Figure 7B & Page 4, Paragraph [0049]), wherein the library stub generator (Figure 3, 315), processes each methods (all public entities) included in the class by generating a stub, wherein Figure 7B, steps 721- 734, teach replacing a reference to software contained in the body of a method of the source code 110, with a stub returned with the appropriate return type value. It should also be noted that this method is performed for all non-private (public) classes. Additionally, any exception is processed in another file, but nonetheless processed, so that the secondary developer can compile class files.

-"*...compiling each of said equivalent public classes; and generating a compilation interface for said first package including each of said compiled equivalent public classes.*" (E.g., see Figure 3 & Page 2, Paragraph [0032]), wherein the secondary developer combines the developers own source code 160, with the equivalent public classes (library stubs 220), compiles it with java compiler 125B, thereby generating a compilation interface 170, for first package 160, wherein the interface 170, receives the compiled source code 160, from the first package and interfaces with the card converter 335B.

2. In regard to claim **7**, the rejections of base claim **1** are incorporated. Furthermore, **Krishna** discloses:

- "*...identifying all public classes included in a Java Archive file.*" (E.g., see Figure 7A, step 704), wherein an IDE file from which to synthesize classes/jar file is disclosed.

3. In regard to claim **10**, the rejections of base claim **1** are incorporated. Furthermore, **Krishna** discloses:

- "*...generating a separate java file for each of said identified public classes.*" (E.g., see Figure 7B, step 738), wherein a separate .java file is generated for each of said identified public classes.

4. In regard to claim **11**, the rejections of base claim **10** are incorporated. Furthermore, **Krishna** discloses:

- "*...compiling each said java file.*" (E.g., see Figure 7B, step 744), wherein a separate .java file is compiled.

5. In regard to claim **12**, the rejections of base claim **11** are incorporated. Furthermore, **Krishna** discloses:

- "*...generating a compilation Java Archive file; and storing each said compiled .java file in said compilation Java Archive file.*" (E.g., see Figure 7B, step 747), wherein the class files are placed in a Java Archive file.

6. In regard to claim **13**, the rejections of base claim **1** are incorporated. Furthermore, **Krishna** discloses:

"*...utilizing said compilation interface within an Integrated Development Environment.*" (E.g., see Figure 1 and Paragraph [0005]), wherein a standard Java development environment is disclosed.

7. As per claims **20** and **23-26**, this is a system version of the claimed method discussed above, in claims **7** and **10-13**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 8), wherein a system employing the above methods is disclosed.

8. As per claims **33** and **36-39**, this is a product version of the claimed method discussed above, in claims **7** and **10-13**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 1 & Paragraph [0053]), wherein a product version of the above methods is disclosed.

Claim Rejections - 35 USC § 103

1. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims **2, 4-6, 9, 15, 17-19, 22, 28, 30-32** and **35** are rejected under 35 U.S.C. 103(a) as being unpatentable over Krishna et al., US 2003/0051233 A1 (hereinafter **Krishna**) in view of Dale Green, "Trail: The Reflection API", The Java Tutorial. Posted November 27th, 1999. Retrieved from <http://Green.com/docs/books/tutorial/reflect/> (hereinafter **Green**).

3. In regard to claim **2**, the rejections of base claim **1** are incorporated. Furthermore, **Green** discloses:

- "*...identifying all entities included in each of said public classes that include a public modifier.*" (E.g., see "Trail: The Reflection API", Page 1), wherein all public class modifiers are discovered along with their fields, methods, superclasses and variables (entities).

Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine a public class modifiers and their entities with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna,** "...only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (E.g., see "Trail: The Reflection API", Page 1).

4. In regard to claim **4**, the rejections of base claim **1** are incorporated. Furthermore, **Green** discloses:

- "*...identifying all public methods included in each of said public classes.*" (E.g., see "Obtaining Method Information", Page 15), wherein one can "...uncover a method's name, return type, parameter types, set of modifiers, and set of throwable exceptions."

Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine public methods modifiers and their entities with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna,** "...only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (E.g., see "Trail: The Reflection API", Page 1, Paragraph 1).

5. In regard to claim **5**, the rejections of base claim **1** are incorporated. Furthermore, **Green** discloses:

- "*...public parameters included in each of said public classes.*" (E.g., see "Obtaining Method Information", Page 15, Paragraph 2), wherein "... the following sample program prints the name, return type, and parameter types of every public method in the Polygon class", wherein the public parameters are included.

Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine public parameters with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna,** "...only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (E.g., see "Trail: The Reflection API", Page **1**, Paragraph 1).

6. In regard to claim **6**, the rejections of base claim **1** are incorporated. Furthermore, **Green** discloses:

- "*...public fields included in each of said public classes.*" (E.g., see "Trail: The Reflection API", Page 1), wherein all public class modifiers are discovered along with their fields, methods and variables (entities).

Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine a public class modifiers and their fields with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna,** "...only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...get information about a class's ...fields..." (Page 1, Paragraph 1).

7. In regard to claim **9**, the rejections of base claim **1** are incorporated. Furthermore, **Green** discloses:

- "*...identifying all public entities included in each of said public classes utilizing Java Reflection.*" (E.g., see "Trail: The Reflection API", Page 1), wherein all public class modifiers are discovered along with their fields, methods and variables (entities) using the Java Reflection API.

Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine Java Reflection with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna,** "...only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (E.g., see "Trail: The Reflection API", Page 1, Paragraph 1).

8. As per claims **15, 17-19** and **22**, this is a system version of the claimed method discussed above, in claims **2, 4-6** and **9**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 8), wherein a system employing the above methods is disclosed.

9. As per claims **28, 30-32** and **35**, this is a product version of the claimed method discussed above, in claims **2, 4-6** and **9**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 1 & Paragraph [0053]), wherein a product version of the above methods is disclosed.

10. Claims **3, 16** and **29** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Krishna** in view of **Green** and further in view of Evans et al., US 6,836,884 B1 (hereinafter **Evans**).

11. In regard to claim **3**, the rejections of base claim **1** are incorporated. But the combined teaching of **Krishna** and **Green** do not expressly disclose, "*...determining whether each of said entities includes a native attribute; in response to a determination that each of said entities includes a native attribute, removing said native attribute from each of said entities; and generating equivalent entities that include no native attributes.*" However, **Evans** discloses:

- "*...determining whether each of said entities includes a native attribute; in response to a determination that each of said entities includes a native attribute, removing said native attribute from each of said entities; and generating equivalent entities that include no native attributes.*" (E.g., see Figure 3 & Column 12, line 63 – Column 13, line 11), wherein native code may be edited from one form to a more general form.

Evans and the combined teaching of **Krishna** and **Green** are analogous art because they are both concerned with the same field of endeavor, namely, compiling software code. Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine replacing native code with the combined teachings method. The motivation to do so, is suggested by **Evans,** "...the user may replace an existing method created in a first source language with a new method created in a second source language.", (E.g., see Column 2, line 65 – Column 3, line 1).

12. As per claim **16**, this is a system version of the claimed method discussed above, in claim **3**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 8), wherein a system employing the above method is disclosed.

13. As per claim **29**, this is a product version of the claimed method discussed above, in claim **3**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 1 & Paragraph [0053]), wherein a product version of the above method is disclosed.

14. Claims **8, 21** and **34** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Krishna** in view of **Green** and further in view of obviousness.

15. In regard to claim **8**, the rejections of base claim **1** are incorporated. But the combined teaching of **Krishna** and **Green** do not expressly disclose, "*...utilizing a java.util.jar utility.*" However, it would have been obvious, to one of ordinary skill in the art to utilize a java.util.jar utility to identify all public classes included in said first package. The motivation to do so is provided by **Green** (E.g., see "Examining Interfaces", Page 1), wherein importing java.util.* is shown, wherein the java.util.jar utility would be included. Furthermore, the java.util.jar is another known method to extract class information and thus it would have been obvious to one of ordinary skill in the art to use a utility already imported to achieve a goal already stated (identify all public classes). Therefore, at the time the invention was made it would have been obvious to identify all public classes included in said first package utilizing a java.util.jar utility with the combined teaching of **Krishna** and **Green**.

16. As per claim **21**, this is a system version of the claimed method discussed above, in claim **8**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 8), wherein a system employing the above method is disclosed.

As per claim **34**, this is a product version of the claimed method discussed above, in claim **8**, wherein all claimed limitations have also been addressed and/or cited as set forth above. For example, see **Krishna** (Figure 1 & Paragraph [0053]), wherein a product version of the above methods is disclosed.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

John Romano

Conferees:

Tuan Q. Dam, SPE 2192 [illegible]

TUAN DAM
SUPERVISORY PATENT EXAMINER

Kakali Chaki, SPE 2193 [illegible]

KAKALI CHAKI
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100